Problem 6 (30 Points)

During the lecture you worked with pipelines in SciKit-Learn to perform feature transformation before classification/regression using a pipeline. In this problem, you will look at another scaling method in a 2D regression context.

You are welcome to use any of the code provided in the lecture activities.

Summary of deliverables:

Sklearn Models (no scaling): Print Train and Test MSE

- Linear Regression (input degree 8 features)
- SVR, C = 1000
- KNN, K = 4
- Random Forest, 100 estimators of max depth 10

Sklearn Pipeline (scaling + model): Print Train and Test MSE

- Linear Regression (input degree 8 features)
- SVR, C = 1000
- KNN, K = 4
- Random Forest, 100 estimators of max depth 10

Plots

- 1x5 subplot showing model predictions on unscaled features, next to ground truth
- 1x5 subplot showing pipeline predictions with features scaled, next to ground truth

Questions

- Respond to the prompts at the end

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
```

```
from sklearn.linear_model import LinearRegression
from sklearn.svm import SVR
from sklearn.ensemble import RandomForestRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.preprocessing import PolynomialFeatures, QuantileTransformer
from sklearn.pipeline import Pipeline
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split

def plot(X, y, title=""):
    plt.scatter(X[:,0],X[:,1],c=y,cmap="jet")
    plt.colorbar(orientation="horizontal")
    plt.xlabel("$x_1$")
    plt.ylabel("$x_2$")
    plt.title(title)
```

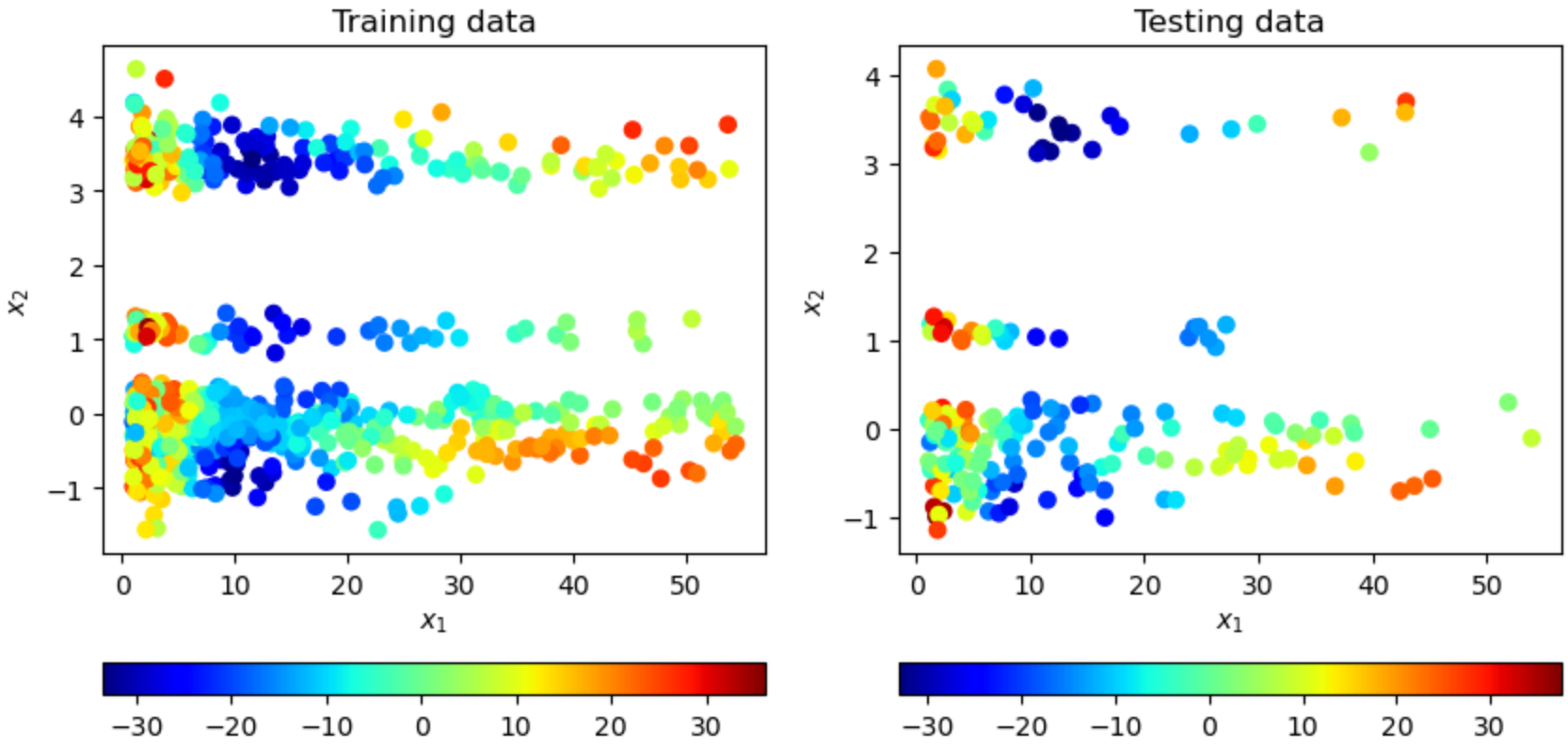
Load the data

Complete the loading process below by inputting the path to the data file "w6-p1-data.npy"

Training data is in `X_train` and `y_train`. Testing data is in `X_test` and `y_test`.

In [2]:
```
# YOUR CODE GOES HERE
# Define path
path = 'data/w6-p1-data.npy'
data = np.load(path)
X, y = data[:,:2], data[:,2]
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=int(0.8*len(y)),random_state=0)

plt.figure(figsize=(10,5))
plt.subplot(1,2,1)
plot(X_train, y_train, "Training data")
plt.subplot(1,2,2)
plot(X_test, y_test, "Testing data")
plt.show()
```



Models (no input scaling)

Fit 4 models to the training data:

- `LinearRegression()`. This should be a pipeline whose first step is `PolynomialFeatures()` with degree 7.
- `SVR()` with C = 1000 and "rbf" kernel
- `KNeighborsRegressor()` using 4 nearest neighbors
- `RandomForestRegressor()` with 100 estimators of max depth 10

Print the Train and Test MSE for each

In [3]:
```
# YOUR CODE GOES HERE

model_names = ["LSR", "SVR", "KNN", "RF"]
```

```
models = [Pipeline([("Polynomial Features", PolynomialFeatures(degree=8)),("Linear Regression", LinearRegression())]),
          SVR(kernel='rbf',C=1000),
          KNeighborsRegressor(n_neighbors = 4),
          RandomForestRegressor(n_estimators=100,max_depth=10)]

all_pred_test = []

for i in range(len(model_names)):
    model = models[i]
    model.fit(X_train,y_train)
    pred_train = model.predict(X_train)
    pred_test = model.predict(X_test)

    print(f'For {model_names[i]} Training MSE is: {mean_squared_error(y_train,pred_train)}')
    print(f'For {model_names[i]} Testing MSE is: {mean_squared_error(y_test,pred_test)} \n')

    all_pred_test.append(pred_test)

all_pred_test = np.array(all_pred_test)
```

```
For LSR Training MSE is: 43.93218438615584
For LSR Testing MSE is: 45.41580195959632

For SVR Training MSE is: 82.04352603565974
For SVR Testing MSE is: 98.63319719407525

For KNN Training MSE is: 26.856498566141628
For KNN Testing MSE is: 47.63617328402055

For RF Training MSE is: 6.127732297000152
For RF Testing MSE is: 25.598864592247963
```

Visualizing the predictions

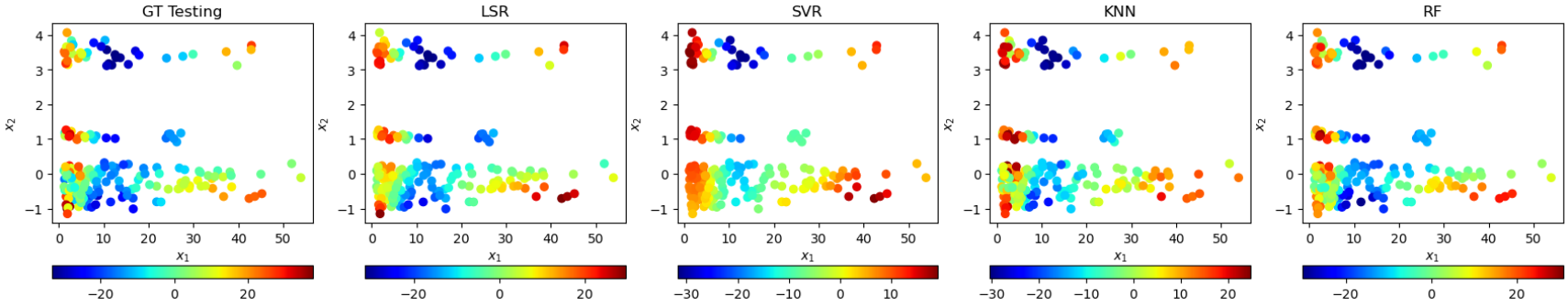
Plot the predictions of each method on the testing data in a 1x5 subplot structure, with the ground truth values as the leftmost subplot.

In [4]:
```
plt.figure(figsize=(21,4))
plt.subplot(1,5,1)
plot(X_test, y_test, "GT Testing")

# YOUR CODE GOES HERE
```

```
for i in range(len(all_pred_test)):
    plt.subplot(1,5,i+2)
    plot(X_test,all_pred_test[i],model_names[i])

plt.show()
```



Quantile Scaling

A `QuantileTransformer()` can transform the input data in a way that attempts to match a given distribution (uniform distribution by default).

- Create a quantile scaler with `n_quantiles = 800`.
- Then, create a pipeline for each of the 4 types of models used earlier
- Fit each pipeline to the training data, and again print the train and test MSE

In [5]:
```
pipeline_names = ["LSR, scaled", "SVR, scaled", "KNN, scaled", "RF, scaled"]

# YOUR CODE GOES HERE
scaler = QuantileTransformer(n_quantiles = 800)

models_scaled = [Pipeline([("Qunatine Scaler",scaler),("Polynomial Features", PolynomialFeatures(degree=8)),
                           ("Linear Regression", LinearRegression())]),
                 Pipeline([("Qunatine Scaler",scaler),("SVR",SVR(kernel='rbf',C=1000))]),
                 Pipeline([("Qunatine Scaler",scaler),("KNeighborsRegressor",KNeighborsRegressor(n_neighbors = 4))]),
                 Pipeline([("Qunatine Scaler",scaler),("RandomForestRegressor",RandomForestRegressor(n_estimators=100,max_depth

all_pred_test_scaled = []

for i in range(len(pipeline_names)):
```

```
    model_scaled = models_scaled[i]
    model_scaled.fit(X_train,y_train)
    pred_train_scaled = model_scaled.predict(X_train)
    pred_test_scaled = model_scaled.predict(X_test)

    print(f'For {pipeline_names[i]} Training MSE is: {mean_squared_error(y_train,pred_train_scaled)}')
    print(f'For {pipeline_names[i]} Testing MSE is: {mean_squared_error(y_test,pred_test_scaled)} \n')

    all_pred_test_scaled.append(pred_test_scaled)

all_pred_test_scaled = np.array(all_pred_test_scaled)
```

```
For LSR, scaled Training MSE is: 38.10305004706555
For LSR, scaled Testing MSE is: 43.18995888435163

For SVR, scaled Training MSE is: 41.03425800595977
For SVR, scaled Testing MSE is: 43.017915737897745

For KNN, scaled Training MSE is: 19.687691313922564
For KNN, scaled Testing MSE is: 36.397038931930005

For RF, scaled Training MSE is: 6.007202973982084
For RF, scaled Testing MSE is: 25.86979036031245
```

Visualization with scaled input

As before, plot the predictions of each *scaled* method on the testing data in a 1x5 subplot structure, with the ground truth values as the leftmost subplot.

This time, for each plot, show the scaled data points instead of the original data. You can do this by calling `.transform()` on your quantile scaler. The scaled points should appear to follow a uniform distribution.

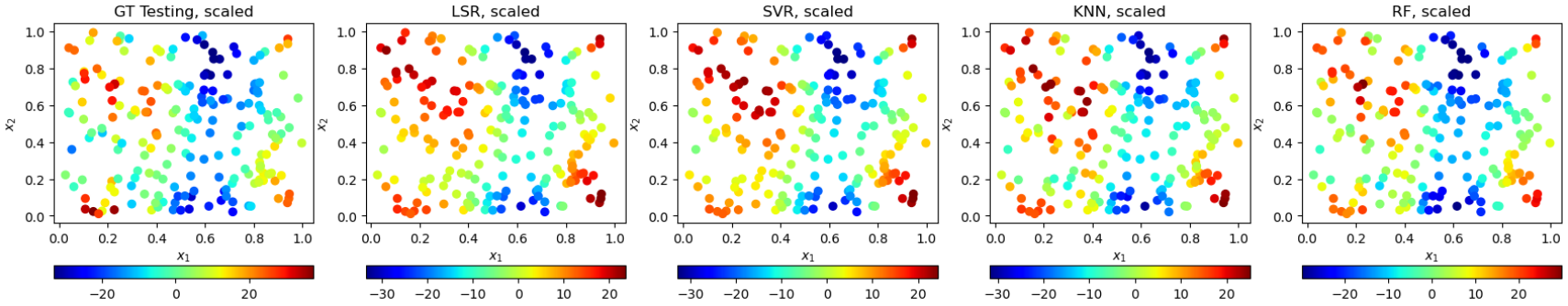
In [6]:
```
# YOUR CODE GOES HERE
X_test_scaled = scaler.transform(X_test)

plt.figure(figsize=(21,4))
plt.subplot(1,5,1)
plot(X_test_scaled, y_test, "GT Testing, scaled")

for i in range(len(all_pred_test_scaled)):
    plt.subplot(1,5,i+2)
```

```
    plot(X_test_scaled,all_pred_test_scaled[i],pipeline_names[i])

plt.show()
```



Questions

1. Without transforming the input data, which model performed the best on test data? What about after scaling?

Without transforming the input data, the Random Forest Regressor performed the best on the test data. After scaling, Random Forest Regressor still performs the best as it gives the lowest MSE.

1. For each method, say whether scaling the input improved or worsened, how extreme the change was, and why you think this is.

The scaling improved for each model. The improvement was the least in Random forest Regressor. The Linear Regression model and KNN had a decent amount of improvement and the SVR model had a huge improvement because of scaling. The reason for Randome forest is because it uses splitting data at various features and also uses randomness. The Linear Regression Model depends on coeffecients and the coeffecients can be very large or small so scaling helps in improving the model. The KNN depends on the distance of nearest features so when the scaling happens the distance can be dominated with the features of larger scale. The SVR highly relies on kernels and the distance between datapoints. The scaling makes the kernel function more reliable.

In [ ]: